CONCEALED BIKE ANTI-THEFT DEVICE

By

Elizabeth Atkinson

Srinidhi Raman

Alex Wen

Final Report for ECE 445, Senior Design, Spring 2022

TA: Hojoon Ryu

5 May 2022

Project No. 45

Abstract

In order to solve the problem of bike theft on campus, we present a concealed bike anti-theft device that both deters potential thieves from stealing the user's bike, and makes it easier for the user to locate the bike if it has been stolen. The device uses a RFID reader to detect the user's RFID card to "unlock" the bike and allow the user to transport the bike easily; if the bike is transported without the user's RFID card, the alarm will sound, alerting passerby to the theft and ideally causing the thief to abandon the bike. In the event the thief does remove the bike from the original location, the device uses a GPS module to determine its location and transmit the location data over a LoRa link to the user base station, where the GPS data is displayed in an easy-to-use GUI for the user to view. Our prototype device is contained in a water bottle that can easily fit into the average bike water-bottle-holder without alerting the thief to the presence of the anti-theft device. The GPS, RFID, alarm, GUI, and microcontroller subsystems are fully functional, and although we were not able to implement the full LoRa link, we were able to identify the most likely causes for this failure.

Contents

1. Introduction

In college campuses around the country, many students have experienced the problem of bike theft. Since bikes are frequently left outside for extended periods of time, they are vulnerable to theft, and unfortunately, once a bike is stolen it is very unlikely to be recovered. In order to solve this problem, a twofold approach is ideal – first, deter the theft itself, and second, improve the likelihood of bicycle recovery after the theft has occurred.

In order to deter potential thieves, we implemented a system to detect when a theft is occurring, and trigger a loud and annoying alarm that will make the thief want to abandon the bike. A RFID "unlocking" system allows the device to differentiate from the bike owner and a bike thief. The bike owner uses their RFID tag to unlock the device, which allows them to ride the bike normally. However, if the bike device detects movement from GPS data without the RFID unlocked, the alarm will sound. The alarm will continue to sound until the GPS no longer detects movement, indicating the thief has abandoned the bike.

For the second component – increasing ease of bicycle recovery after a theft – the device uses the GPS module to track its location over time and transmit the data over a LoRa link to the user base station. The user's base station will receive the data and display it for the user to view. This way, the user will be able to find their bike after it has been stolen. Since LoRa is a very low-power protocol with an extremely long range even in urban environments, it will enable the user to track their bike even after it has been moved relatively far away from its original location. Additionally, since LoRa operates in the 900-MHz ISM band, it requires no radio licensing to operate, therefore greatly simplifying the deployment of the device.

2 Design

2.1 Block Diagram

The diagram below shows the conceptual block diagram for the device.

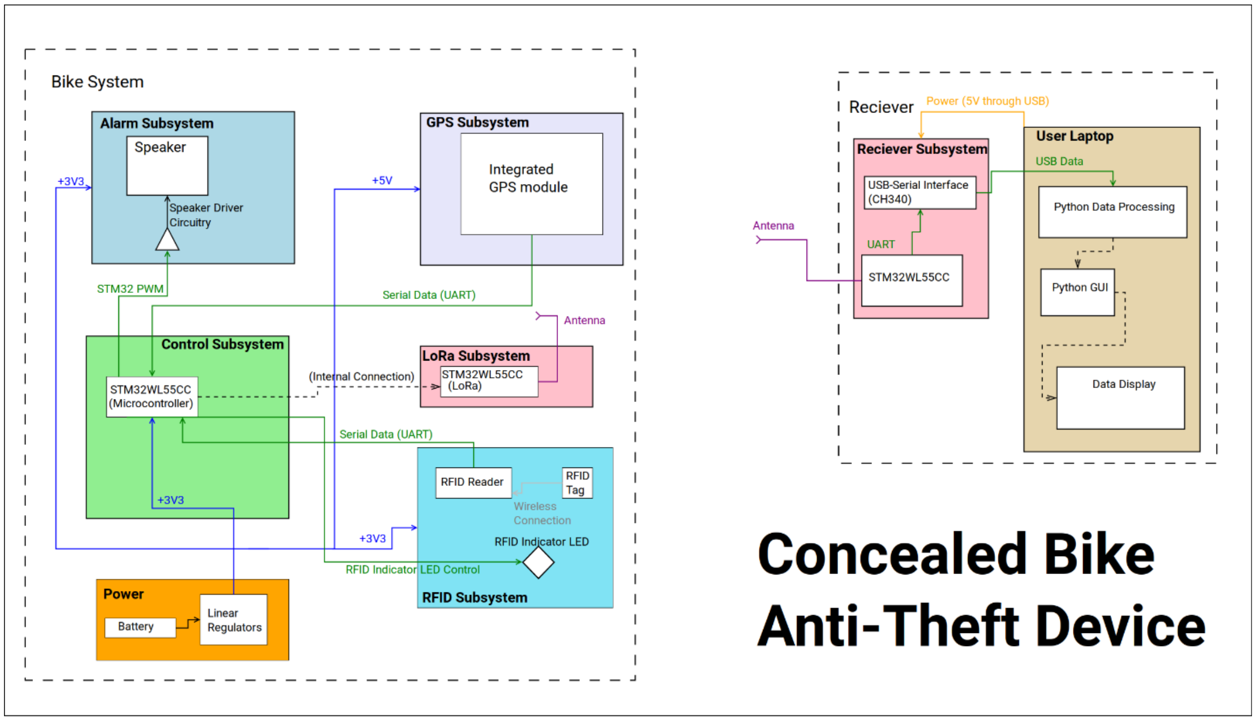


Figure 1: System Block Diagram

2.2 Control Subsystem

Design Procedure:

For our control unit, we selected the STM32WL55CC microcontroller. We selected this microcontroller primarily due to the integrated SubGHz radio, which is much more convenient for implementing the radio subsystem than an external LoRa IC. Although there are many integrated LoRa modules commercially available, we chose to use the STM32WL since we would likely learn more by implementing the radio ourselves rather than using a commercial module. The dual core ARM Cortex-M0/Cortex-M4 provided adequate computing power and peripherals (two UARTs and several GPIOs) which were necessary for our device.

Design Details:

Our control system successfully implemented a state-based control system shown in figure A.4 in Appendix 1. The full circuit diagram is shown in figure A.1. We used STM32CubeMX to generate peripheral initialization code, and for debugging, we used an STLINK probe with Black Magic Probe firmware running in self-hosted mode.

2.3 LoRa Subsystem

Design Procedure:

As discussed above, we used the STM32WL55CC integrated microcontroller/SubGHz receiver for our project. Our LoRa subsystem used the SubGHz receiver which includes hardware to implement the LoRa modulation scheme. LoRa is a low-power, spread-spectrum modulation scheme which allows the user to trade receiver sensitivity for data rate; for our low-data rate, low-power, high-range system, it was an ideal choice. STMicroelectronics also provides a firmware package for this microcontroller which includes a LoRaWAN upper-level protocol stack, which we initially planned to use as well. However, since we do not actually need the MAC and upper-level stack functionality for our system, we switched to only using the low-level hardware radio driver instead to minimize the complexity of the firmware.

Design Details:

We used the ST-provided RF design guide to design the matching network in between the RF output of the MCU and an external antenna [2]. Since the RF output is a class-D amplifier, the matching network was quite complex since it included all the pulse-shaping, matching and filtering circuitry necessary to match the device to a 50-ohm output. The full circuit is shown in figure A.2.

In order to implement the radio in firmware, we used the low-level hardware radio driver provided by ST. This driver essentially provided access to the internal SPI interface between the STM32 microcontroller and SubGHz radio peripheral. In order to test our device, we initialized the radio, set the frequency and power, set the RF switch control lines, and then sent the CW test command. Although we were able to verify that the firmware did give the correct RF output with the STM32WL55 Nucleo dev board, our PCB did not have any RF output; refer to section 3.2 for more on this topic.

2.4 GPS Subsystem

Design Procedure:

We originally planned to use the EM-506 GPS module from Sparkfun, but we discovered during testing that the UART output uses 0 stop bits which is impossible to receive using any other UART device. So, we switched to the Adafruit Ultimate GPS module, which uses a very standard UART configuration (9600 8N1), and were able to communicate with this module easily. One other concern in this subsystem was the distance calculation, which is somewhat slow since the microcontroller does not have an FPU. In order to prevent the calculation from slowing down the microcontroller processing excessively, we decreased the frequency of the GPS updates and instead only requested data from the GPS every 10 seconds. We used the STM32 timer peripheral to implement the 10 second timer to avoid using (blocking) delay functions in the code.

Design Details:

Due to the complexity of implementing a GPS receiver from scratch, we used a commercial GPS module, so there was very little circuit design for this subsystem. In firmware, we implemented NMEA message parsing that put data from the UART peripheral into a struct that contained each field of the NMEA message. To calculate the distance between two GPS locations, we used the flat-surface approximation [1]:

$$D = R\sqrt{(\Delta\phi)^2 + (\cos(\phi_m)\,\Delta\lambda)^2}$$

Where R is the radius of the earth, ϕ denotes latitude, λ denotes longitude, and $\phi_m = (\phi_1 + \phi_2)/2$. In order to determine whether the device was in motion or stationary, we calculated the distance between two subsequent GPS location measurements (which are each 10 seconds apart as explained above). If the distance exceeded the setpoint of 8 meters, the device would consider itself "in motion". The threshold of 8 meters was chosen due to the fact that experimentally, the "drift" of the GPS measurement when the device was stationary tended to be below 5 meters in most cases, so a threshold of 8 meters decreased the likelihood of false positives for motion.

Timing the GPS data operations was critical to our project, since the state transitions depended on knowing the elapsed time between different GPS locations. Since the HAL_Delay functions simply block any code from executing until the time has elapsed, we avoided these functions, since it is important that our state machine continues to execute during the time in between GPS data operations. Instead, we polled one of the STM32 timer peripherals to keep track of the elapsed time between the previous GPS operation and the current time. When the time exceeded the set point (in this case 10 seconds), the device polled the GPS module again.

2.5 RFID Subsystem

Design Procedure:

We designed this subsystem around the main component, the ID-3LA RFID Module from Sparkfun. This module was chosen as it could communicate with the STM32 through UART, and noted that it had a 30 cm detection range when using a suitable antenna. We originally planned to create an antenna by winding a coil, but decided to use a general 1 mH antenna for convenience and accuracy.

Design Details:

The circuit was designed to include an LED as visual indicator, and appropriate connections for each of its pins according to details given by the datasheet for the RFID module.

2.6 Alarm Subsystem

Design Procedure:

Originally, we intended to use a Piezo buzzer due to its small size which would be easy to conceal on the bike device. However, the Piezo buzzer we selected was not loud enough, so we switched to a normal speaker instead. Since a critical requirement of our project is a loud and annoying noise to scare away a potential bike thief, it was essential that our alarm subsystem was able to be loud. The speaker was slightly larger, but significantly louder than the piezo buzzer, so we decided to use it for the alarm subsystem.

Design Details:

We used one of the STM32 timer peripherals to generate a PWM control signal at the desired frequency of 5 kHz whenever the alarm was activated. Since the STM32 cannot provide enough current to drive the speaker directly, we used the simple circuit below as the speaker driver. The BJT acts as a switch to turn the speaker on and off at the signal frequency. This circuit allowed the speaker to make a loud and annoying tone when the alarm was activated. Since we did not have time to order an additional PCB, we implemented the driver circuit on protoboard for our demo.

For our demo, we added a timeout to the ALARM state so that after 5 seconds it would automatically transition back to the IDLE state – this was primarily because the alarm did end up being very loud and annoying to listen to for more than 5 seconds at a time. However, in a real device, we would only allow the ALARM state to time out after the device has remained in a stationary location for longer than 5 minutes. This would incentivize the bike thief to abandon the bike rather than continue stealing it, since the alarm is quite painful to listen to.

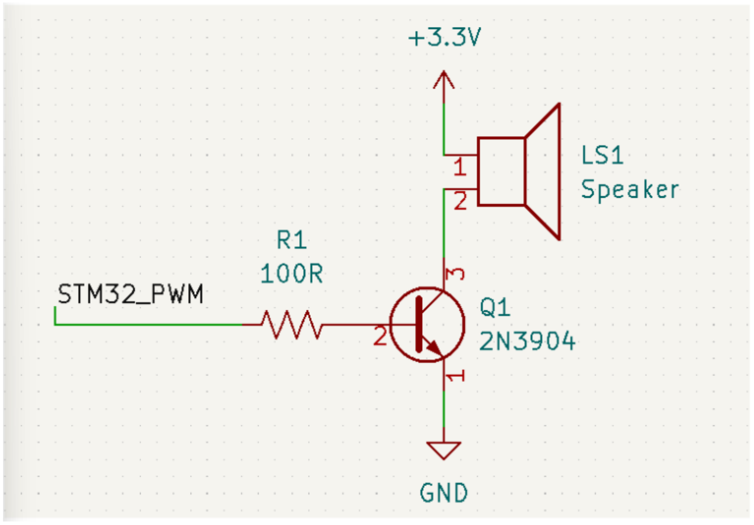


Figure 2: Speaker Driver Circuit.

Similarly to the GPS subsystem, we polled the STM32 timer peripheral to keep track of the time elapsed between the speaker starting to sound and the current time in order to implement the 5 second timeout. This again enabled our state machine to continue executing rather than blocking code execution for the duration of the timeout.

2.7 GUI Subsystem

Design Procedure:

The Python Graphical User Interface (GUI) is used to provide a visual representation of data – GPS location – obtained from the USB serial port such that the user can interact with data in various ways, including analyzing and understanding the history of where the bike has been and provide a mapping view of current location based on current GPS data. Originally, this was to be done from an FTDI board with an I2C link for data transmission; however, we were not able to obtain a working FTDI chip, thus data transmission was conducted using USB serial. In terms of working with the source code, the transition in the data transmission methodology simplified the way we obtain data as it merely requires the internal serial module to access the serial port from the USB device.

Design Details:

In the Python code, the interaction between the USB serial and the GUI is the internal serial module. From this module, the device can be read based on its identifier – in our case, it's along the lines of '/dev/ttyUSB0'. The data obtained from the device can then be read through the various read functions, depending on the objective of the function. For the GUI itself, the windows application is built using Python's internal pyqt5 module. The module provides for the general application overlay, followed by a series of buttons – as well as an internal signal connection that displays a new window when clicked. As of the formulation of this report, the GUI can obtain and retain the data in static state – that is, it can show a certain amount of data as predefined in the source code; it has yet to be able to constantly poll the device and obtain and show data continuously. Also, the integration of mapping feature has not been successful.

2.8 Power Subsystem

Design Procedure:

Since the device is meant to be mobile and able to be carried on a bike, it was essential that the device is battery powered. We originally planned to use four AA batteries in series, but eventually decided on a single 9V battery instead since it was difficult to design a battery pack on a short schedule. We used linear regulators to regulate the 9V input down to 5V for the GPS board and 3V3 for the MCU and all other peripherals.

Design Details:

The main board also contains the 3V3 and 5V linear regulators, and all of the peripherals are powered from this board. To further conserve power, we included load switches on all the power supplies for the GPS, RFID and speaker module so that the subsystems could be fully turned off when not in use. The load switches functioned correctly and turned off the power to each subsystem when switched off by the microcontroller control signal; however, we did not end up implementing the power-saving measures in firmware since we did not have time. Additionally, the power saving measures would have made it much more challenging to demo the device, so we did not include them in the demo.

3. Design Verification

3.1 Control Subsystem

Overview:

Since our control system simply ran the state machine, our verification was not very complex; much of the functionality was demonstrated during the demo. Our full requirements and verification are below.

Requirement	Verification	Outcome

1. The microcontroller implements a state-based control system with power-saving measures in the idle/safe states**.**	1. Flash the firmware to the device. 2. Monitor the state transitions during device operation.	**Success:** We verified that the MCU behaves according to our desired state machine.
1. The microcontroller packetizes data to be sent over LoRa at the update rate specified by the state diagram in figure 3.	1. Flash test firmware that cycles through each of the states. 2. Verify that the TX update rate is greater than once per hour in the IDLE, UNLOCK, and TRANSIT states.	**Partial Success:** Our microcontroller was able to create data packets from the GPS data, but since our LoRa subsystem was not functional, we were unable to transmit the packets.

3.2 LoRa Subsystem

Overview:

Unfortunately, our LoRa subsystem was the main failure in the device. We verified that the firmware test program using the low-level radio driver worked on the Nucleo dev board, but it did not work on our PCB, leading us to conclude there was a hardware issue with our board. We could not fully diagnose the issue due to the limited time and resources in the course, but we do have several ideas for what could have caused the issue.

First, we encountered several issues with the oscillator. We initially used a 32 MHz SMD external crystal since the RF transceiver requires relatively high frequency accuracy; due to a mistake in the board layout, we could not use the oscillator we ordered, and instead had to bodge in a through-hole crystal. It is likely that the through-hole crystal soldered onto the pads introduced enough parasitic inductance and capacitances to cause the internal PLL to fail. This hypothesis is supported by the fact that when we polled the radio error state, it returned intermittent PLL lock errors. To further investigate this issue, we would try correcting the layout issue and assembling a new board with good crystal trace routing; however, we did not have time to do this in this class.

Our second possibility was a separate generic hardware fault and/or an issue with the internal PA. Since the internal PA is delicate and usually requires a dedicated matching network to prevent signal reflections from damaging the device, it is possible that we inadvertently destroyed the PA output stage over the course of testing the device. Unfortunately, since we did not have enough time or materials to assemble a second board to test, we could not test a fresh board to determine if we had damaged the PA in our original board. The least likely option in our view is the a matching network erorr, since we followed the matching network design guide in detail to design our matching network. However, since we could not find a VNA to test the matching network, we could not verify the input impedance and it is very possible we simply presented the PA with a VSWR that exceeded the damage rating and permanently destroyed the PA.

Requirement	Verification	Outcome

1. The LoRa subsystem matching network and antenna have a VSWR of less than 10 at the RF output of the STM32WL55CC.	1. Populate the RF matching network before populating the microcontroller. 2. Calibrate the VNA from 800-1200 MHz with PCB TRL standards. 3. Terminate the SMA output of the PCB with a 50 ohm load. 4. Measure the input impedance looking into the RF matching network at the RF output of the STM32WL55CC. 5. Calculate the VSWR from the measured input impedance and the specified PA output impedance in the STM32 datasheet.	**Could not verify:** We did not have access to a VNA, so we were unable to test or verify this requirement.
1. The antenna is hidden in a common bike component such as a reflector so that it is not visibly obvious.	1. Objective visual verification. 2. Ask other senior design students.	**Unsuccessful:** We did not use an antenna since our LoRa link did not work, so we did not satisfy this requirement.

3.3 GPS Subsystem

Overview:

Due to the complexity of implementing GPS from scratch, we used a commercially-available GPS module, the Adafruit Ultimate GPS V3. Since we used a module we did not design any hardware for this subsystem, but we wrote the firmware to communicate with the microcontroller over UART, parse messages, and calculate the distance between two GPS readings.

Requirement	**Verification**	**Outcome**
1. The GPS module can communicate with the microcontroller at a update rate of greater than 1 Hz.	1. Flash a test program to the microcontroller that continuously polls the GPS module. 2. Verify that the GPS module transfers GPS data at an update rate of greater than 1 Hz.	**Success:** We were able to communicate with the GPS module at a rate of greater than 1 Hz. However, in our final design, we did decrease this update rate to once every 10 seconds as discussed in section 2.4.

3.4 RFID Subsystem

Overview:

Requirement	**Verification**	**Outcome**

1. The RFID detector will detect the user’s RFID tag within 10 seconds.	1. Place the RFID tag within 10 cm of the RFID receiver. 2. Monitor the RFID output data to verify it has detected the RFID card within 10 seconds.	**Success.**
1The RFID subsystem will provide a visual indication that the RFID tag has been detected within 1 second of detecting the RFID tag.	1. Place the RFID tag within 10 cm of the RFID receiver. 2. Monitor the RFID reader output data and the RFID LED to verify that it turns on within 10 seconds of detecting the RFID tag.	**Success.**
1. The RFID subsystem triggers the alarm if movement occurs and the user’s RFID is not detected.	1. Ensure that the RFID tag is out of range (> 1 m). 2. Move the bike 5m from the original location to verify the alarm is triggered	**Success.**
1. The RFID indicator LED is concealed in an unobtru-sive location on the bike.	1. Visual verification.	**Unsuccessful:** We did not finish integrating the device into the bike, so the LED was not concealed.

3.5 GUI Subsystem

Overview:

Our GUI verification process was somewhat limited due to the fact that we did not have a functioning LoRa link. Additionally, we changed our final design to use the CH340 IC rather than the I2C/USB FTDI IC we had originally planned to use. However, we were still able to verify much of the functionality of the GUI itself.

Requirement	Verification	Outcome
1. The receiver board will in-terface with the computer over USB.	1. Flash a test program to the base station board that sends continuous data over the I2C/USB interface. 2. Connect the base station board to the user computer and view the data sent over USB. 3. Verify the received data matches the sent data.	**Success:** We were able to communicate data to the computer over USB, however, we did modify the design to use the CH340 UART/USB converter IC rather than the FTDI I2C/USB IC.

1. The receiver board will re-ceive LoRa packets from the bike-based device.	1. Flash a test program to the TX (bike system) board that transmits continuous data. 2. Flash a test program to the RX (base station) board that listens for data and continuously sends received data to the computer over I2C (USB to the computer). 3. Verify the data sent from the TX board is received by the RX board.	**Partial Success:** As discussed before, we were not able to implement the LoRa link. However, we were able to receive "dummy" messages from the board that mimicked the type of message that would be received over LoRa from the bike-based device.
1. The GUI will display GPS data over the past day, week, month, or year on a map	1. Flash a test program to the TX (bike system) board that transmits data with (false) timestamps that cover a 6-month period. 2. Transmit the data and receive it with the base station board. 3. Using the GUI, plot the data and verify the output plots match the data sent by the TX board	**Partial Success:** The GUI was able to display data, but we did not fully finish the map functionality.
1. The GUI will allow the user to view plots of the average speed and distance traveled.	1. Flash a test program to the TX (bike system) board that transmits data with (false) timestamps that cover a 6-month period. 2. Transmit the data and receive it with the base station board. 3. Using the GUI, plot the average speed and distance traveled and verify that it matches the data transmitted by the bike system board.	**Unsuccessful:** We did not have time to implement this functionality.

3.6 Power Subsystem

Overview:

In order to enable our device to function for extended periods of time in a self-contained manner, we powered the device using a battery, and included power-saving measures such as load switches to fully turn subsystems on and off.

Requirement	Verification	Outcome

1. The power system will provide 3V3 +/- 0.4 mV to all subsystems that require 3V3, and 5V to the subsystems that require 5V.	1. A multimeter will be used to ensure the voltage and current are as specified at different points on the board.	**Success:** All subsystems are powered correctly.
1. The power system will power the device for over 24 hours.	1. Connect a multimeter/oscilloscope to measure the voltage at the battery terminals. 2. Record data every two hours over two 12 hour periods. 3. The data from the device will be read to ensure that the values were reasonably constant over time.	**Unsuccessful:** Due to time constraints, we did not test this functionality.
1. The microcontroller can fully power off the GPS and RFID subsystems when not in use by turning off the load switches.	1. Flash a test program to the micrcontroller that toggles the control signals GPS_ PWR and RFID_ PWR. 2. Measure the output voltage on pin 1 of U7 and U8 to verify that the RFID and GPS subsystems are powered off.	**Success:** The load switches were able to turn off power to each of the subsystems and were able to be controlled by the microcontroller.

4. Costs

4.1 Parts

Part	Manufacturer	Retail Cost ($)	Bulk Purchase Cost ($)	Actual Cost ($)
ID-3LA	Sparkfun	27.95	27.95	27.95
Adafruit Ultimate GPS	Adafruit	29.95	29.95	0 (already had)
STM32WL55CC	STMicroelectronics	12.94	11.69	12.94
RFID Tag	Sparkfun	2.10	2.10	2.10
Speaker (SP-6619)	Soberton	3.60	2.98	0 (already had)
Miscellaneous components	Digikey	~100	~100	~100
Total				**142.99**

4.2 Labor

For each engineer:

$42.07/hour * 2.5 * 120 hours = $12,621

Total for three engineers:

3*12,621=$37,863

5. Conclusion

5.1 Accomplishments

As shown in the Requirements and Verifications sections, we were successfully able to implement near-complete functionality of the RFID, GPS, and Alarm subsystem, and we made significant strides towards implementing the GUI on the receiver device. Although we were not able to implement the LoRa link, we learned how to use the complex ST LoRaWAN firmware, and in particular, how to use the low-level radio driver to implement LoRa directly to avoid the complexity of the upper-level protocol stack. We were able to demonstrate transmission using the Nucleo dev board, which suggests that with hardware corrections, a new revision of our PCB would have a high likelihood of successfully transmitting LoRa.

5.2 Uncertainties

By far our biggest uncertainty was the unsuccessful LoRa subsystem, and unfortunately, we are still not sure what caused the issue. However, if we were to continue the project, we have a clear path forward for continuing to diagnose the problem and eventually coming to a conclusion about how to solve the issue.

In addition to the LoRa subsystem, our final device was unfortunately not very polished. Due to time constraints, we used a water bottle to contain our final model, but it was not very well "concealed". In a future revision of this project, we would work to make a more polished and unobtrusive packaging method for the device so that it could be concealed in a real-life water bottle holder without looking suspicious.

5.3 Ethical considerations

This project is designed with IEEE's code of ethics in mind, and the main point as we work on the project is "to hold paramount the safety, health and welfare of the public, to strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment."

The design is implemented with LoRa and RF implementation, which involves the transmission and reception of information wirelessly. As such, an ethical consideration that is paramount to our objective is that data is received accurately. This is important because if given false or failed readings thieves can make off with the bike even with the flawed design attached due to failure or misreading of information. As such, the RFID tag communicates with the transmitter such that the alarm will sound if and only if the bike is in motion and reaches out of range of the RFID.

5.4 Future work

Since our main failure with this project was the LoRa subsystem, the main goal of future work would be to successfully implement that subsystem. With a new board revision, we could fix the issues of the oscillator footprint and possible matching network issues and transmit with the board. Once the hardware issues are fixed, it would also be important to develop a LoRa transmission protocol that is robust and can deal with multiple bikes with the anti-theft device on them. For this task, we would need to investigate the use of existing protocols such as LoRaWAN and assess whether they would be adequate or whether it would be more effective to create a new, likely simpler protocol for our use case.

References

[1] *Geographical distance,* Wikipedia, 2022. Available at: https://en.wikipedia.org/wiki/Geographical_distance.

[2] STMicroelectronics, “RF matching network design guide for STM32WL Series”, AN5457, 2020.

Appendix A Detailed Circuit Schematics

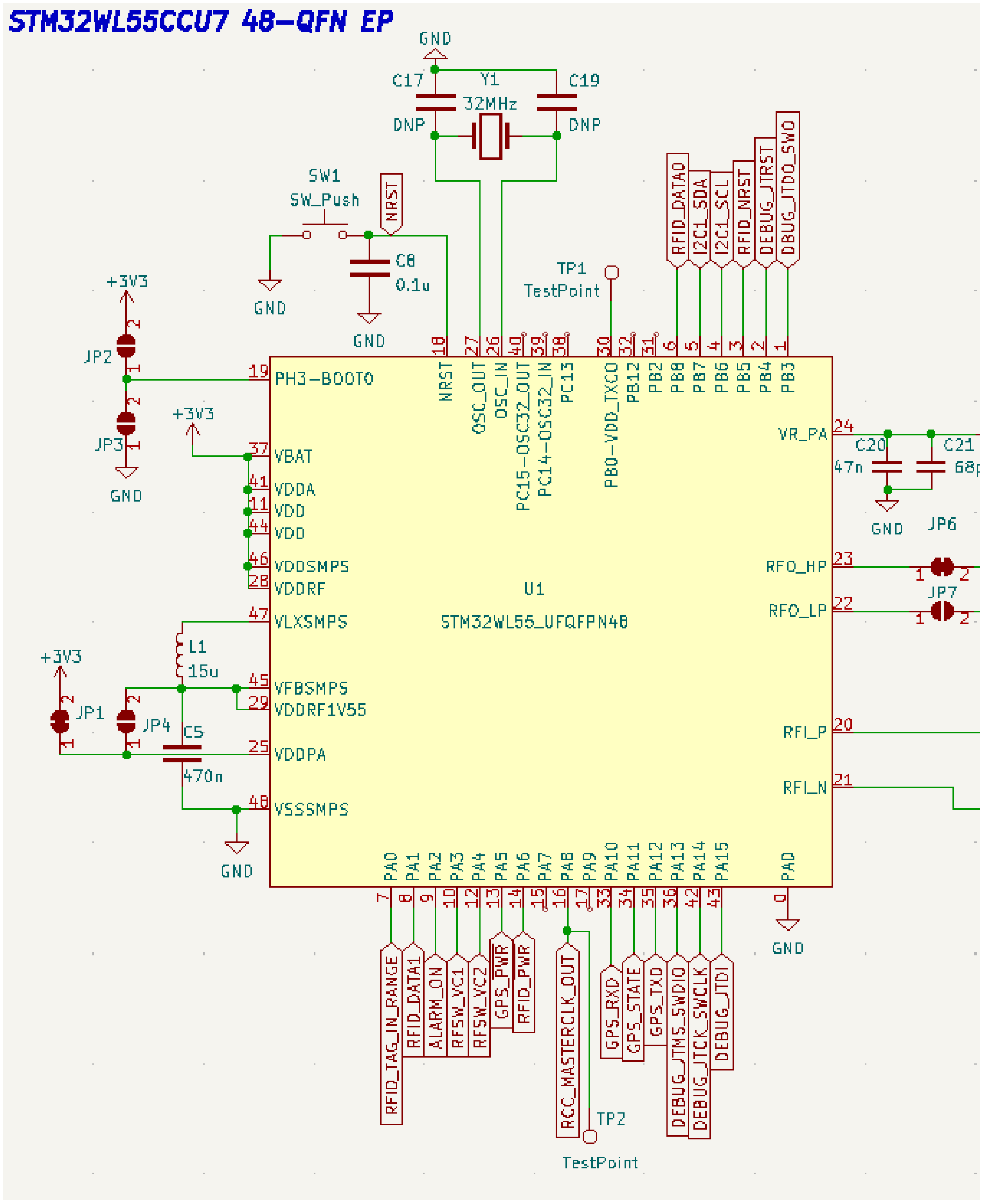


Figure A.1: Microcontroller Subsystem Schematic

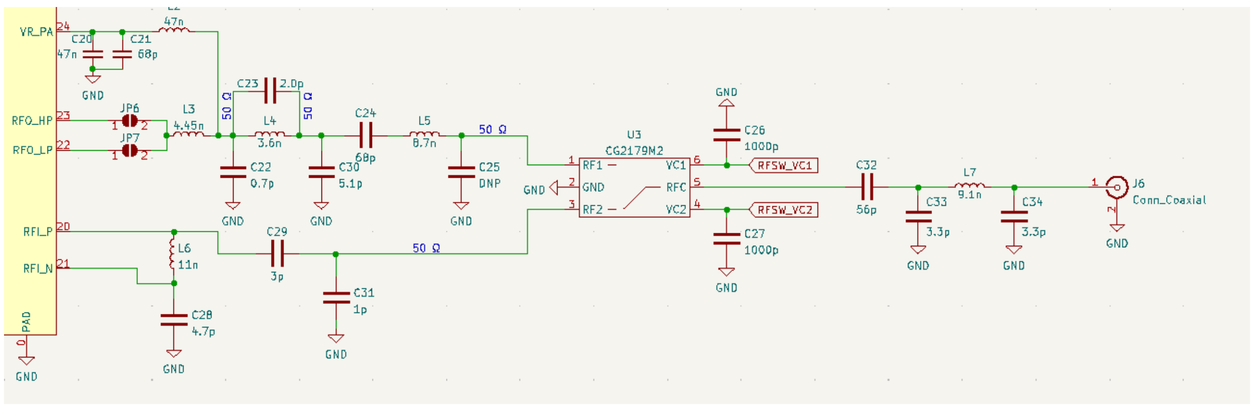


Figure A.2: RF Matching Network Schematic

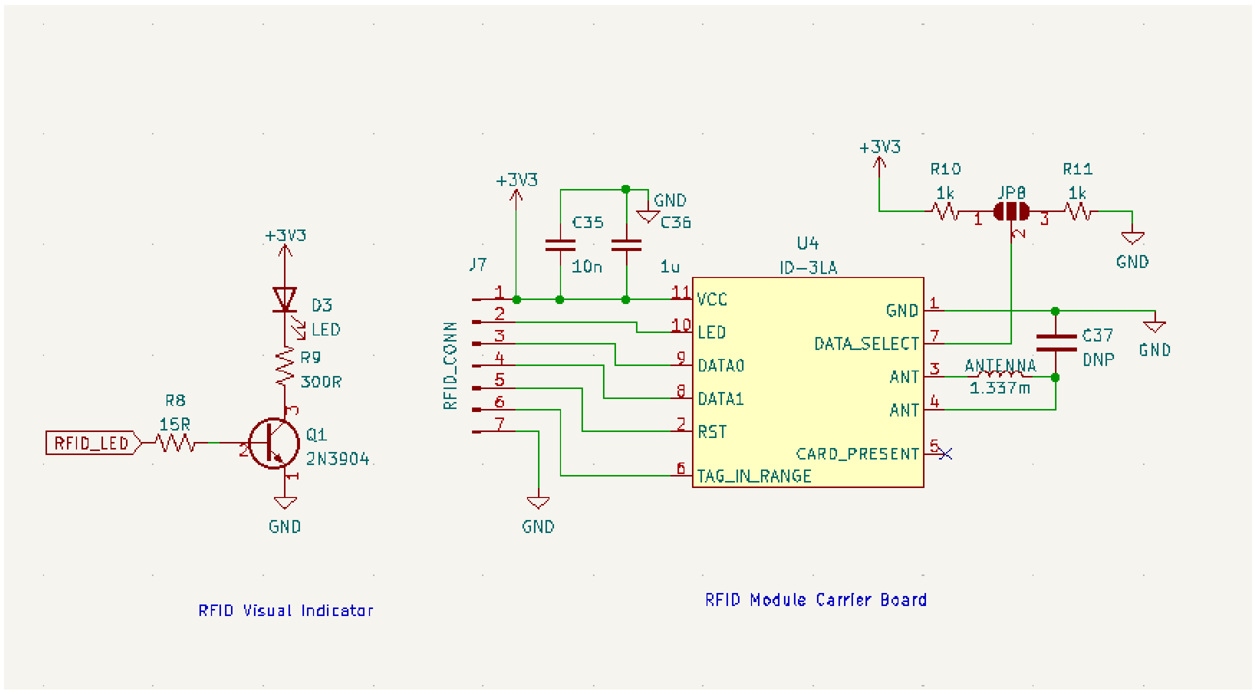
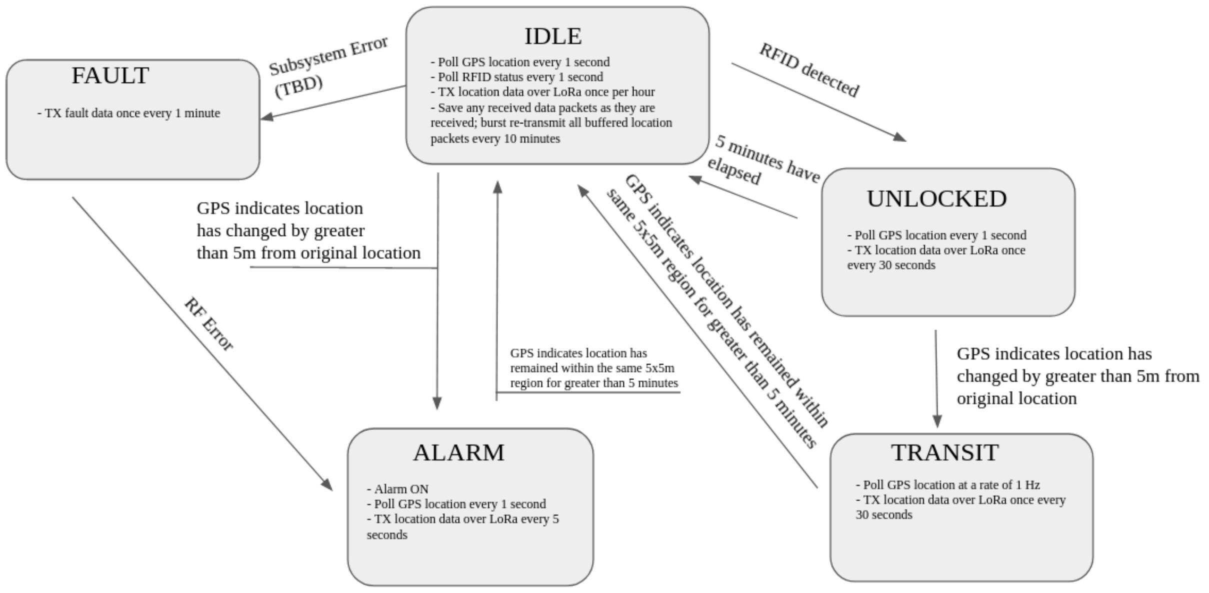


Figure A.3: RFID Carrier Board Schematic



IDLE
- Poll GPS location every 1 second
- Poll RFID status every 1 second
- TX location data over LoRa once per hour
- Save any received data packets as they are received; burst re-transmit all buffered location packets every 10 minutes
FAULT
- TX fault data once every 1 minute
Subsystem Error (TBD)
RFID detected
5 minutes have elapsed
UNLOCKED
- Poll GPS location every 1 second
- TX location data over LoRa once every 30 seconds
GPS indicates location has changed by greater than 5m from original location
RF Error
GPS indicates location has remained within the same 5x5m region for greater than 5 minutes
GPS indicates location has remained within same 5x5m region for greater than 5 minutes
GPS indicates location has changed by greater than 5m from original location
ALARM
- Alarm ON
- Poll GPS location every 1 second
- TX location data over LoRa every 5 seconds
TRANSIT
- Poll GPS location at a rate of 1 Hz
- TX location data over LoRa once every 30 seconds

Figure A.4: System State Diagram

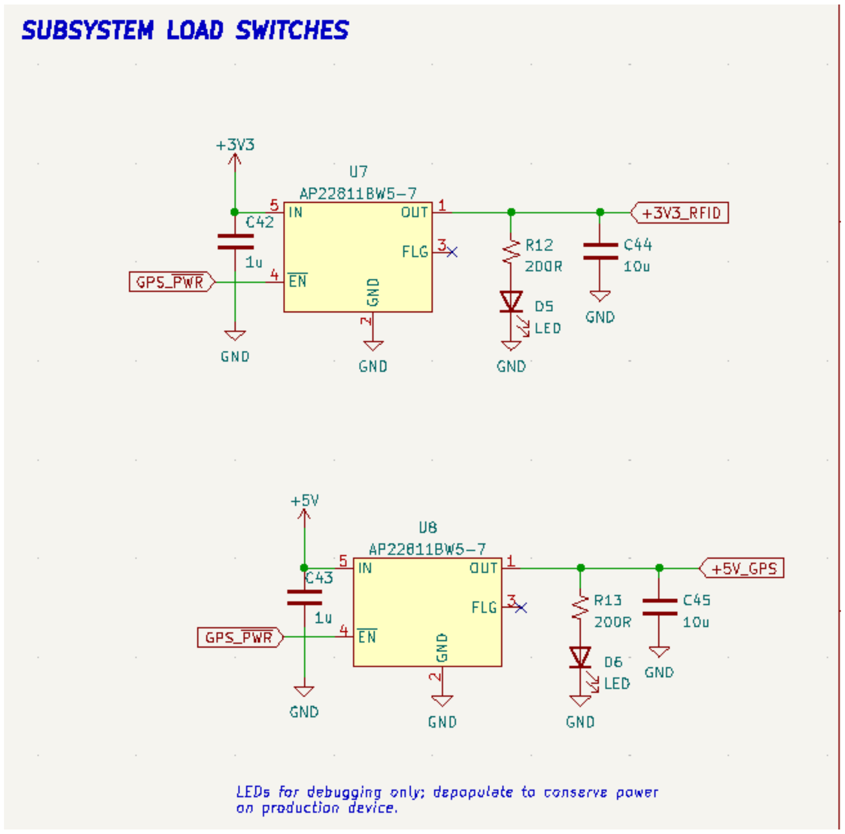


Figure A.5: Subsystem Load Switches.